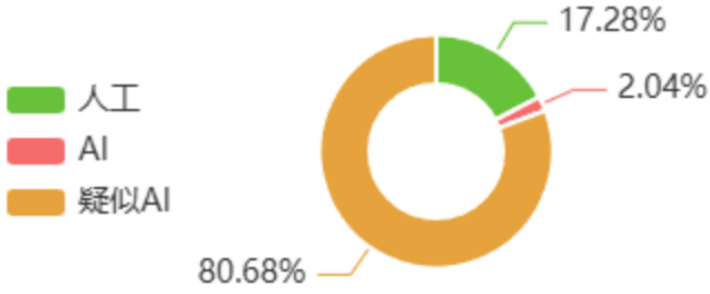
朱雀AI生成检测报告单

检测时间：2025/7/26 22:09:03

检测结果

嗅探到AI浓度：**82.71%**



AI分布图

人工 (AIGC值: 0-0.5)　疑似AI (AIGC值: 0.5-0.85)　AI (AIGC值: 0.85-1)

片段解析

序号	片段	占全文比例	AIGC值	疑似程度
1	片段1	3.42%	97.55%	
2	片段2	14.80%	24.57%	
3	片段3	32.67%	89.22%	
4	片段4	2.55%	44.63%	
5	片段5	44.50%	81.73%	

6	片段6	2.05%	100.00%	

检测片段详情

NO. 1 片段1 AIGC值 97.55%

GPG（GNU Privacy Guard）是一个完全免费的加密软件，它实现了 OpenPGP 标准。简单来说，GPG 就像是一个数字保险箱，可以帮助你：

🔒 加密文件和邮件，让只有特定的人才能查看
✅ 验证文件的完整性，确保内容没有被篡改
✍️ 为你的身份提供数字签名，证明某个文件确实是你创建的
GPG 使用的是非对称加密技术，也就是我们常说的"公钥加密"。这种技术的巧妙之处在于，你有两把钥匙：一把公钥可以公开分享给任何人，一把私钥必须严格保密。别人用你的公钥加密的信息，只有你的私钥才能解开；而你用私钥签名的文件，别人可以用你的公钥来验证确实是你本人所为。

想象一下，你有一个透明的盒子（公钥）和一把只有你才有的钥匙（私钥）。任何人都可以往盒子里放东西并锁上，但只有你能用你的钥匙打开。

PGP：GPG 的前身 📜
在了解 GPG 之前，我们需要先认识它的"父亲"——PGP（Pretty Good Privacy）。

PGP 诞生于 1991 年，由菲尔·齐默曼（Phil Zimmermann）开发。当时互联网刚刚兴起，人们开始意识到数字通信的隐私保护问题。齐默曼开发 PGP 的初衷很简单：让普通人也能拥有军用级别的加密保护。

PGP 一经推出就引起了巨大反响，甚至惊动了美国政府。

NO. 2 片段2 AIGC值 24.57%

因为当时美国法律将强加密技术视为军用物资，禁止出口。齐默曼因此还面临了刑事调查，不过最终政府撤销了指控。

随着时间的推移，PGP 的发展出现了分歧。商业版本成为了付费软件，而开源社区则需要一个免费的替代品。1997 年，GNU 项目启动了 GPG 的开发，目标是创建一个完全兼容 PGP 但完全免费的实现。

可以说，GPG 继承了 PGP 的技术精髓，但去掉了商业限制，让加密技术真正普及到了每一个人。

在 Linux 上使用 GPG 🐧
安装 GPG
大多数 Linux 发行版都预装了 GPG。如果没有，可以通过包管理器安装：

```
# Ubuntu/Debian
sudo apt update
sudo apt install gnupg2

# CentOS/RHEL
sudo yum install gnupg2

# Fedora
sudo dnf install gnupg2

# Arch Linux
sudo pacman -S gnupg
```

验证安装
安装完成后，可以检查版本：

gpg --version

你应该看到类似这样的输出：

gpg (GnuPG) 2.2.40
libgcrypt 1.10.1
Copyright (C) 2022 g10 Code GmbH
License GNU GPL-3.0-or-later
<https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
...

在 Windows 上使用 GPG 🪟

安装选项

Windows 用户有几种选择：

Gpg4win（推荐）🌟：这是最流行的 Windows GPG 发行版，包含图形界面

下载地址：https://www.gpg4win.org/

包含了 GPG、图形界面工具 Kleopatra 等

Git for Windows 🐙：如果你已经安装了 Git，通常已经包含了 GPG

可以在 Git Bash 中直接使用 gpg 命令

Windows Subsystem for Linux (WSL) 🔧：在 WSL 中安装 Linux 版本的 GPG

基本概念详解 🔔

当你查看密钥列表时，会看到一堆看起来很复杂的输出信息。别慌！我们先来理解一下这些输出到底是什么意思 😊

/root/.gnupg/pubring.kbx
----
pub   rsa4096 2025-07-26 [SC] [有效至：2030-07-25]

pub   rsa4096 2025-07-26 [SC] [有效至：2030-07-25]
      4BA8 1EB7 BDCC 0428 C3FD 8A2A 5E83 E6FE A940 F75E
uid   [绝对] Liu Zhen <noc@932.moe>
sig 3   5E83E6FEA940F75E 2025-07-26 Liu Zhen <noc@932.moe>
sub   rsa4096 2025-07-26 [E] [有效至：2030-07-25]
      6D9C EDEE A650 07BE 48D1 A798 09D7 A3C6 273C A832
sig   5E83E6FEA940F75E 2025-07-26 Liu Zhen <noc@932.moe>

密钥类型 🔑

每个密钥前面都有几个字母：pub/sub/sec/ssb，它们代表什么呢？

pub 和 sec：主密钥的公钥和私钥
sub 和 ssb：子密钥的公钥和私钥

为什么需要子密钥？ 🤔

想想看，如果你只有一把万能钥匙，丢了就全完蛋了 😱 但如果你有一把主钥匙放在保险箱里，平时用几把副钥匙，即使副钥匙丢了，也只需要废掉这把副钥匙，重新配一把就行！

这就是子密钥的妙用：

主密钥安全存放，只在必要时使用 🏦
日常操作用子密钥，丢了也不怕 💪
每个子密钥还可以用不同的加密算法，灵活性更强 ✨

密钥用途 🎯

每个密钥后面的方括号里有字母，代表这把密钥能干什么：

E：加密（Encryption） 📨
S：签名（Signing） ✍️
C：认证其他密钥（Certification） ✅
A：身份认证，比如 SSH 登录（Authentication） 🚪

默认生成的密钥通常是：主密钥 [SC] + 子密钥 [E]，也就是主密钥负责签名和认证，子密钥负责加密 📦

加密算法 🔢
rsa3072 这样的标识表示用的是 3072 位的 RSA 算法。就像锁的复杂程度，位数越高越安全，但速度会慢一些 ⚖️

想要其他算法？用 gpg --full-gen-key 命令可以自定义选择！

指纹 👆
那一长串看起来像乱码的字符其实是密钥的"指纹"，就像人的指纹一样，每个密钥都有独一无二的标识 🆔

User ID 👤
格式是：真实姓名 (备注) <邮箱>，用来标识密钥的主人是谁。

有效性 ✅
每个用户 ID 后面的方括号显示有效性状态：

[unknown]：不知道是不是真的这个人 🤷
[ultimate]：绝对信任（通常是自己的密钥）💯
怎么验证有效性？🔍
面对面交换：最安全
验证指纹：通过电话等可靠方式确认指纹是否正确 📞
确认后，用自己的密钥给对方签名，表示"我认可这个人" ✨

信任网络 🕸️
没法亲自验证每个人？没关系！可以利用信任网络 😊

比如你信任总统，总统认证了国务卿，那你也可以间接信任国务卿。

NO. 3 片段3 AIGC值 89.22%

这就像朋友的朋友也可能是朋友一样！

信任等级：

ultimate：终极信任（自己）👑
full：完全信任 💯
marginal：部分信任（需要 3 个这样的人都认证才算数）🤝
never：不信任 ❌
自签名和交叉签名 🔄
为了防止坏人冒充，所有的用户 ID 和子密钥都需要主密钥的「自签名」，就像盖章确认「这确实是我的」 📋

撤销 🚫
如果密钥不想用了，不能简单删除，而是要"撤销"。就像在报纸上登声明"此密钥作废"一样 📰

撤销的密钥还能解密和验证以前的信息，只是不能用于新操作了。

有效期 ⏰
可以给密钥设置过期时间，不是为了定期换密钥，而是提醒别人及时更新公钥信息 📅

过期的密钥也还能解密和验证旧信息，只是会提醒"已过期" ⚠️

选择密钥 🎯
有很多方式可以指定要用哪个密钥：

指纹：40 个字符，最准确但最长 📏
短 ID：后 8 位（不推荐，不安全） ⚠️
长 ID：后 16 位（推荐）✅
邮箱：<xxx@xxx> 或 @xxx 📧
姓名：部分匹配即可，不区分大小写 👤
特殊技巧：在 ID 后加 ! 可以强制使用特定密钥，不让 GPG 自动选择 💪

怎么样，是不是感觉 GPG 没那么可怕了？记住，这些都是为了保护你的数据安全，多花点时间理解是值得的！ 🛡️ ✨

基本操作详解 🛠️

1. 生成密钥对 🔑

第一步是生成你的公钥和私钥对。这就像是制作一把独一无二的钥匙和锁：

```
gpg --full-generate-key
```

让我们逐步看看整个过程：

步骤 1：选择密钥类型

```
Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1
```

直接按回车选择默认的 RSA。

步骤 2：选择密钥长度

```
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (3072) 4096
```

输入 4096 获得更强的安全性。

步骤 3：设置有效期

```
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
```

<n>m = key expires in n months
<n>y = key expires in n years
Key is valid for? (0) 5y
建议设置 2~5 年有效期，输入 5y。这个为主密钥有效期，我们可以之后设置有效期更短的子密钥用于各种用途。

步骤 4：确认信息

Key expires at 2030年07月25日 星期四 20时24分31秒 CST
Is this correct? (y/N) y
步骤 5：输入个人信息

Real name: Liu Zhen
Email address: noc@932.moe
Comment:
You selected this USER-ID:
"Liu Zhen <noc@932.moe>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
步骤 6：设置密码

系统会弹出对话框要求你设置密码。这个密码用来保护你的私钥，请设置一个强密码并牢记！

步骤 7：生成随机数

We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.

这时你可以随意敲击键盘、移动鼠标来帮助生成随机数。整个过程可能需要几分钟。

2. 查看密钥 👀

生成完成后，可以查看你的密钥：

```
# 查看公钥
gpg --list-keys
# 或者 gpg -k
```

输出示例：

```
/root/.gnupg/pubring.kbx
------------------------
pub   rsa4096 2025-07-26 [SC] [有效至：2030-07-25]
      4BA81EB7BDCC0428C3FD8A2A5E83E6FEA940F75E
uid             [ 绝对 ] Liu Zhen <noc@932.moe>
sub   rsa4096 2025-07-26 [E] [有效至：2030-07-25]
```

让我解释一下这些信息：

- pub：主公钥
- rsa4096：RSA 算法，4096 位长度
- 2025-07-26：创建日期
- [SC]：S=签名，C=认证功能
- 4BA81EB7...：密钥指纹（唯一标识）
- uid：用户 ID
- [ultimate]：信任级别
- sub：子密钥，用于加密（E=加密）

查看私钥：

```
gpg --list-secret-keys
```

3. 导出公钥 📤

要与他人交换加密信息，我们需要将公钥分享给对方。这就像把你的"透明盒子"给别人，让他们能往里面放东西：

导出为 ASCII 格式
推荐，可以通过邮件发送
gpg --armor --export noc@932.moe > noc_public.asc
让我们看看生成的文件：

cat noc_public.asc
ASCII 格式的公钥看起来像这样：

-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGiEya8BEACwalEFlkFPpiFlpvcoaLHa+eHTpOlmhfVSw5+JDebKES1cHaJy
pSR1VtHKWvUjFtJSZo8Kk2AgV5mPh1iOiw+A0/H1RoJDCoHnlXrqFytcsfrsdn5N
...
-----END PGP PUBLIC KEY BLOCK-----
我们也可以导出为二进制格式（文件更小）：

gpg --export noc@932.moe > noc_public.gpg
实际示例：分享公钥给朋友

假设我需要把公钥发送给朋友阳帆：

导出公钥：
gpg --armor --export noc@932.moe > noc_public.asc
通过邮件或聊天工具发送这个文件给阳帆

告诉阳帆我的密钥指纹，让他验证：

gpg --fingerprint noc@932.moe
输出：

pub   rsa4096 2025-07-26 [SC] [有效至：2030-07-25]
      4BA8 1EB7 BDCC 0428 C3FD 8A2A 5E83 E6FE A940 F75E
uid   [绝对] Liu Zhen <noc@932.moe>
sub   rsa4096 2025-07-26 [E] [有效至：2030-07-25]

4. 导入他人的公钥 📥

当我们收到别人的公钥时，需要导入到我们的密钥环中。这就像收到别人的"透明盒子"：

gpg --import friend_public.asc

你会看到类似输出：

gpg: key 87654321ABCDEF12: public key "Li Si <lisi@example.com>" imported
gpg: Total number processed: 1
gpg: imported: 1

验证密钥指纹：

gpg --fingerprint lisi@example.com

通过安全渠道（电话、面对面等）与李四确认指纹是否正确

如果指纹正确，可以设置信任级别：

gpg --edit-key lisi@example.com

在 GPG 交互界面中输入：

gpg> trust

选择信任级别（通常选择 4 = 完全信任）：

Please decide how far you trust this user to correctly verify other users' keys
(by looking at passports, checking fingerprints from different sources, etc.)

  1 = I don't know or won't say

2 = I do NOT trust
3 = I trust marginally
4 = I trust fully
5 = I trust ultimately
m = back to the main menu

Your decision? 4
输入 quit 退出。

5. 上传公钥到密钥服务器 🌐
为了方便他人获取你的公钥，可以将其上传到公共密钥服务器：

首先获取你的密钥 ID
gpg --list-keys zhangsan@example.com
找到长串的密钥指纹，比如
1A2B3C4D5E6F7890ABCDEF1234567890FEDCBA98

上传到默认服务器
gpg --send-keys
1A2B3C4D5E6F7890ABCDEF1234567890FEDCBA98
或者指定特定服务器：

gpg --keyserver hkps://keys.openpgp.org --send-keys
1A2B3C4D5E6F7890ABCDEF1234567890FEDCBA98
常用的密钥服务器包括：

🔗 hkps://keys.openpgp.org（推荐）
🔗 hkps://keyserver.ubuntu.com
🔗 https://keys.xmsl.im
🔗 hkps://pgp.mit.edu（不推荐）
6. 从密钥服务器下载公钥 🔍
如果你知道某人的邮箱或密钥 ID，可以直接从服务器下载：

```
# 通过邮箱搜索
gpg --keyserver hkps://keys.xmsl.im --search-keys im@932.moe
```

你会看到搜索结果：

```
gpg: data source: https://104.21.64.133:443
(1)     Liu Zhen <im@932.moe>
          263 bit EDDSA key EA88B86666666666, 创建于：2024-04-10
Keys 1-1 of 1 for "im@932.moe". 输入数字以选择，输入 N 翻页，输入 Q 退出 > Q
```

输入 1 下载第一个密钥，输入 Q 退出选择。

或者直接通过密钥 ID 下载：

```
# 直接通过密钥 ID 下载我的测试密钥
gpg --recv-keys EA88B86666666666
```

7. 加密文件 🔒

现在来到最重要的部分——加密文件。这就像把秘密放进朋友的"透明盒子"里：

基础加密示例：

假设你有一个文件 secret.txt，内容是：

```
今晚的聚会地点改到了星巴克
密码是：coffee123
```

为李四加密这个文件：

```
gpg --encrypt --armor --recipient lisi@example.com secret.txt
```

这会生成 secret.txt.asc 文件，内容看起来像：

```
-----BEGIN PGP MESSAGE-----
```

hQIMAwnXo8YnPKgyARAAmmAHZ76lcfDoxHydoT3g6SCVrKM9
Nd6F5uENPKiBTXDh
（一大串看起来像乱码的字符）
...
-----END PGP MESSAGE-----
同时给多个人加密：

gpg --encrypt --armor --recipient lisi@example.com --recipient wangwu@example.com secret.txt
这样李四和王五都能解密这个文件。

NO. 4 片段4 AIGC值 44.63%

8. 解密文件 🔓
收到加密文件后，解密很简单：

基础解密：

gpg --decrypt secret.txt.asc > decrypted_secret.txt
系统会提示输入私钥密码：

gpg: 由 4096 位的 RSA 密钥加密，标识为 09D7A3C6273CA832，生成于 2025-07-26
      “Liu Zhen <noc@932.moe>”
Passphrase: [输入你的私钥密码]
直接查看内容：

gpg --decrypt secret.txt.asc
输出：

gpg: 由 4096 位的 RSA 密钥加密，标识为 09D7A3C6273CA832

生成于 2025-07-26
    “Liu Zhen <noc@932.moe>”
今晚的聚会地点改到了星巴克
密码是：coffee123
9. 数字签名 ✍️
数字签名可以证明文件确实是你创建的，且没有被篡改。

NO. 5 片段5 AIGC值 81.73%

这就像在文件上盖你的"专属印章"：

创建分离式签名（推荐）
假设你有一个重要文档 report.pdf：

```
gpg --detach-sign --armor report.pdf
```

这会生成 report.pdf.asc 签名文件。现在你有两个文件：

report.pdf（原文件）
report.pdf.asc（签名文件）
创建内嵌签名

```
gpg --sign --armor document.txt
```

这会生成 document.txt.asc，包含原文和签名在一个文件里。

同时加密和签名

```
gpg --encrypt --sign --armor --recipient lisi@example.com
important_document.txt
```

这样生成的文件既加密了（只有李四能看），又有你的签名（李四知道确实是你发的）。

10. 验证签名 ✅
收到签名文件后，验证签名：

验证分离式签名

gpg --verify report.pdf.asc report.pdf

如果签名有效，你会看到：

gpg: Signature made 2024年1月15日 星期一 14时30分25秒 CST
gpg: using RSA key
1A2B3C4D5E6F7890ABCDEF1234567890FEDCBA98
gpg: Good signature from "张三 (工作用密钥)
<zhangsan@example.com>" [ultimate]

如果文件被篡改，你会看到：

gpg: BAD signature from "张三 (工作用密钥)
<zhangsan@example.com>" [ultimate]

验证内嵌签名并提取原文

gpg --decrypt document.txt.asc > original_document.txt

实际应用场景 💼

场景一：安全发送敏感文档 📊

情况：你需要给同事发送包含敏感信息的财务报表。

步骤：

获取同事的公钥：

gpg --search-keys colleague@company.com
或者让同事直接发给你公钥文件
gpg --import colleague_public.asc

验证公钥指纹（通过电话或面对面确认）：

gpg --fingerprint colleague@company.com

加密并签名文档：

gpg --encrypt --sign --armor --recipient
colleague@company.com financial_report.xlsx

通过邮件发送：

发送加密后的文件 financial_report.xlsx.asc
在邮件中可以写：

"附件是 2077 Q3 财务报表的加密版本，请用你的 GPG 私钥解密。"
同事收到后解密：
gpg --decrypt financial_report.xlsx.asc --output financial_report.xlsx
场景二：团队协作中的安全通信 👥
情况：团队需要安全地共享项目信息

设置过程：

团队成员交换公钥：
每个人导出自己的公钥
gpg --armor --export alice@team.com > alice_public.asc
gpg --armor --export bob@team.com > bob_public.asc
gpg --armor --export charlie@team.com > charlie_public.asc

互相导入
gpg --import alice_public.asc
gpg --import bob_public.asc
gpg --import charlie_public.asc
验证所有人的指纹（通过安全渠道）

日常使用

发送给整个团队的加密消息
gpg --encrypt --armor \
  --recipient alice@team.com \
  --recipient bob@team.com \
  --recipient charlie@team.com \
  team_meeting_notes.txt

签名重要决策文档
gpg --sign --armor project_requirements.pdf

高级技巧和最佳实践 🚀

创建撤销证书

```
# 在密钥被泄露时用于废除密钥
gpg --gen-revoke zhangsan@example.com > revoke_cert.asc
# 将此文件安全保存!
```

子密钥管理

```
# 添加新的子密钥
gpg --edit-key zhangsan@example.com
gpg> addkey
# 选择用途（签名/加密/认证）

# 设置子密钥过期时间不同于主密钥
gpg> key 1
gpg> expire
```

备份密钥

```
# 导出私钥（极其重要！）
gpg --armor --export-secret-keys zhangsan@example.com >
private_key_backup.asc
# 将此文件安全保存，最好离线存储
```

常见问题解答 ❓

Q: 我忘记了私钥密码怎么办？ 🤔

A: 很遗憾，没有任何方法可以恢复忘记的私钥密码。这也是 GPG 安全性的体现。你需要：

生成新的密钥对：
gpg --gen-key
如果你之前创建了撤销证书，使用它废除旧密钥：
gpg --import revoke_cert.asc
gpg --send-keys OLD_KEY_ID # 发布撤销信息
通知所有联系人更新你的公钥

预防措施：

使用密码管理器存储 GPG 密码

定期测试密码是否还记得
考虑使用较短（12~32 位）但足够安全的密码
Q: 可以同时拥有多个密钥对吗？ 🔑
A: 当然可以！这是很常见的做法：

```
# 为不同用途生成不同密钥
gpg --gen-key  # 工作用密钥
gpg --gen-key  # 个人用密钥
```

使用场景：

🏢 工作邮箱一个密钥
📧 个人邮箱一个密钥
🔧 特定项目一个密钥
切换使用：

```
# 指定特定密钥加密
gpg --encrypt --local-user work@company.com --recipient friend@example.com file.txt
```

Q: GPG 和 PGP 文件格式兼容吗？ 🔄
A: 是的，GPG 完全兼容 PGP 的文件格式：

用 GPG 加密的文件可以用 PGP 解密
用 PGP 加密的文件可以用 GPG 解密
密钥格式也完全兼容
Q: 密钥长度越长越安全吗？ 📏
A: 一般来说是的，但要考虑实际需要：

推荐配置：

🔐 4096 位 RSA：当前最佳选择，安全性和性能平衡
🚀 3072 位 RSA：也很安全，性能更好
⚡ 2048 位 RSA：最低建议，适合老设备
性能对比示例：